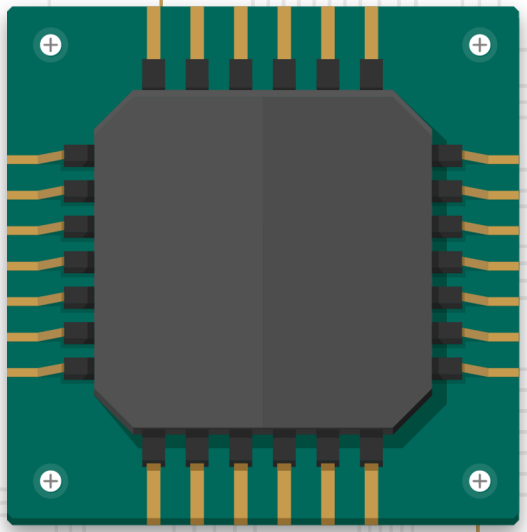
Ottimizzazione MPI di un recommendation system



Source:
https://github.com/ibalejandro/MPIrecommendationsystem

Progetto del corso di Calcolo Parallelo ed ad Alte Prestazioni

Melillo Achille	399000541
Nicoletti Ermanno	399000540
Perugini Pio Antonio	399000545

Recommendation system

Software in grado di creare delle raccomandazioni personalizzate specifiche per l'utente così da aiutarlo nelle sue scelte.

Tipologia di software di successo in diversi ambiti, tra cui l'e-commerce e lo streaming multimediale.

Tutti i principali player online in questi settori hanno almeno un sistema di raccomandazione, che alimentano il modello di business **long tail**.

Recommendation system

Content based
Basato sulla somiglianza tra gli articoli e le preferenze dell'utente.
Rimane nella confort zone dell'utente, non «azzarda» a qualcosa di nuovo.

Filtro collaborativo
Utilizza azioni di utenti simili per suggerire nuovi elementi.
Soffre del «cold start» per utenti nuovi o con gusti unici e può essere influenzato dalla manipolazione dei voti.

Ibrido
Combina entrambi per migliorare la precisione e la diversità delle raccomandazioni, ma richiede uno sforzo di sviluppo maggiore.

Recommendation system

Filtro collaborativo
Utilizza azioni di utenti simili per suggerire nuovi elementi. Soffre del «cold start» per utenti nuovi o con gusti unici e può essere influenzato dalla manipolazione dei voti.

È l'approccio scelto dal sistema preso in analisi.

Codice sviluppato in C++ e già parallelizzato utilizzando le librerie MPI.

NETFLIX amazon

Spotify Booking.com

ebay YouTube

Recommendation system

Generazione della «Utility Matrix»

È matrice utente-item, il cui valore **i,j** è la valutazione reale che l'utente **i** ha rilasciato per l'item **j**.

È l'unica informazione che alimenta il sistema di raccomandazione.

Tramite lo snippet a destra (*genRandomMatUI.cpp*), generiamo una matrice randomica, utilizzata come unico dataset per le misurazioni successive.

Come da documentazione rilasciata nel repository:

./genRandomMatUI 100 100 matrixUI.txt

Tuttavia, il codice indicato, non prevede il salvataggio su file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

int main(int argc, char **argv) {
  srand (time(NULL));

  if (argc != 3) return 0;

  int movies = atoi(argv[1]);
  int users = atoi(argv[2]);

  for (int i = 0; i < movies; ++i) {
    for (int j = 0; j < users; j++) {
      int rating = rand() % 6;
      printf("%d ", rating);
    }
    printf("\n");
  }

  return 0;
}
```

Recommendation system

Generazione della «Utility Matrix»

Come da documentazione rilasciata nel repository:

./genRandomMatUI 100 100 matrixUI.txt

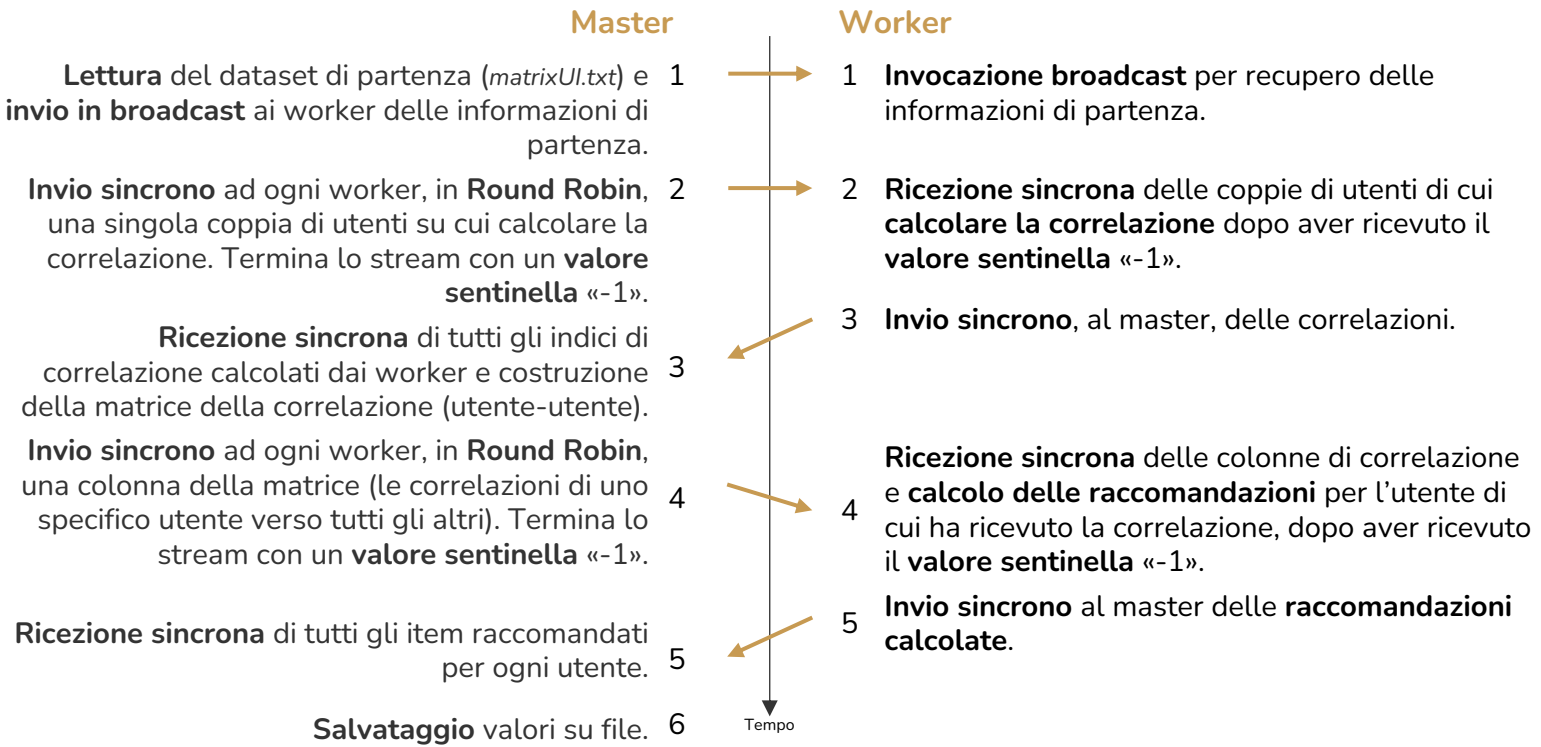
Tuttavia, il codice indicato, non prevede il salvataggio su file.

È stato quindi modificato il codice per salvare la matrice su file, in modo da poterla usare per i test futuri.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void generaMatrice(int righe, int colonne, const char *nomeFile) {
    // Apri il file in modalità scrittura
    FILE *file = fopen(nomeFile, "w");

    // Verifica se il file è stato aperto correttamente
    if (file == NULL) {
        fprintf(stderr, "Errore nell'apertura del file.\n");
        exit(EXIT_FAILURE);
    }

    // Inizializza il generatore di numeri casuali con il tempo attuale
    srand((unsigned int)time(NULL));

    // Genera e salva la matrice
    for (int i = 0; i < righe; i++) {
        for (int j = 0; j < colonne; j++) {
            int valore = rand() % 6; // Numeri casuali compresi tra 0 e 99
            fprintf(file, "%d ", valore);
        }
        fprintf(file, "\n");
    }

    // Chiudi il file
    fclose(file);
}

int main(int argc, char *argv[]) {
    // Verifica che siano stati passati 3 argomenti da riga di comando
    if (argc != 4) {
        fprintf(stderr, "Utilizzo: %s <numero righe> <numero colonne> <nome file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Ottieni il numero di righe, colonne e il nome del file dai parametri
    int righe = atoi(argv[1]);
    int colonne = atoi(argv[2]);
    const char *nomeFile = argv[3];

    // Verifica che il numero di righe e colonne sia positivo
    if (righe <= 0 || colonne <= 0) {
        fprintf(stderr, "Il numero di righe e colonne deve essere positivo.\n");
        exit(EXIT_FAILURE);
    }

    // Chiama la funzione per generare e salvare la matrice
    generaMatrice(righe, colonne, nomeFile);

    printf("Matrice generata e salvata con successo nel file %s.\n", nomeFile);

    return 0;
}
```

Workflow

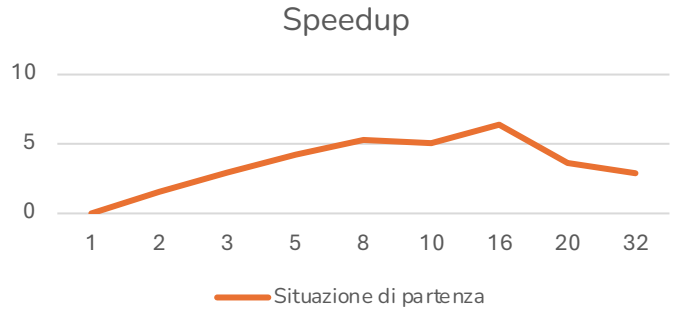
Computazione di tipo **Master-Worker**
Informazioni di partenza: numero degli utenti, numero degli item, numero di raccomandazioni per utente desiderate

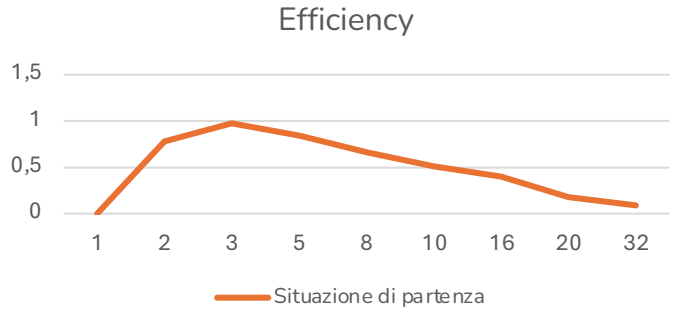


Situazione iniziale

Aspetti **critici**:
- numero di messaggi scambiati **elevato**;
- comunicazione **sincrona** dove **non necessaria**.

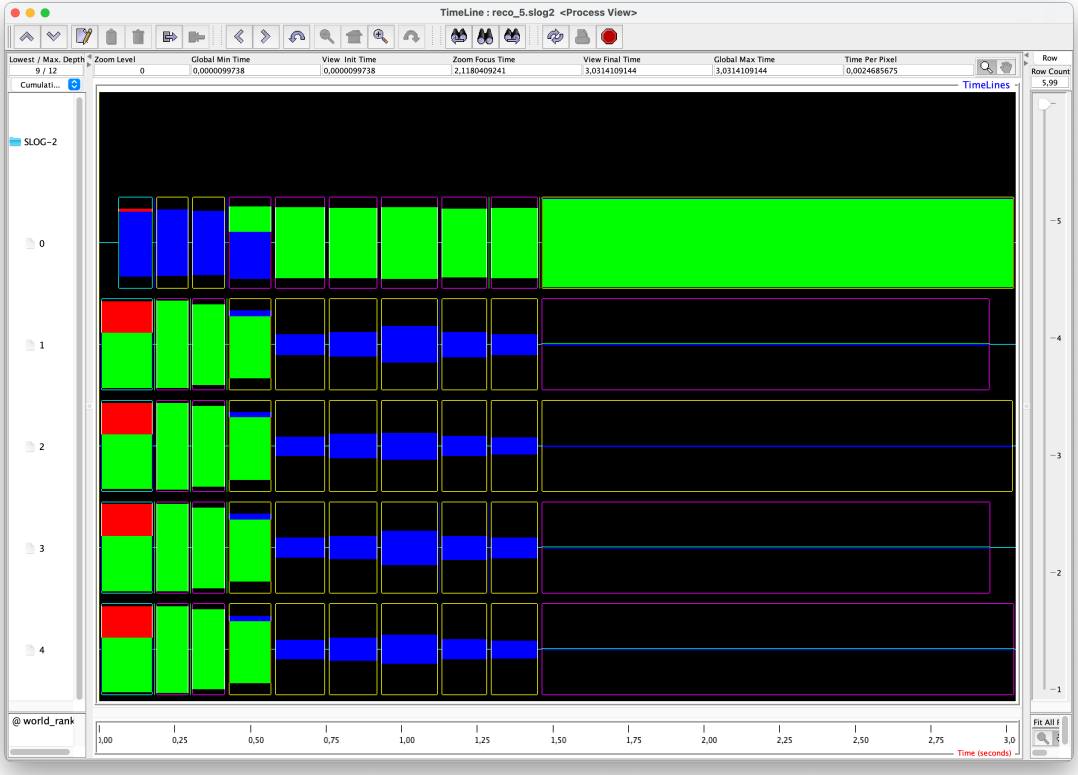
Situazione di partenza					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,554	0,777	9,58
1	3	3	2,925	0,975	5,09
1	5	5	4,218	0,844	3,53
1	8	8	5,280	0,660	2,82
1	10	10	5,082	0,508	2,93
1	16	16	6,391	0,399	2,33
2	20	20	3,623	0,181	4,11
2	32	32	2,908	0,091	5,12





Situazione iniziale

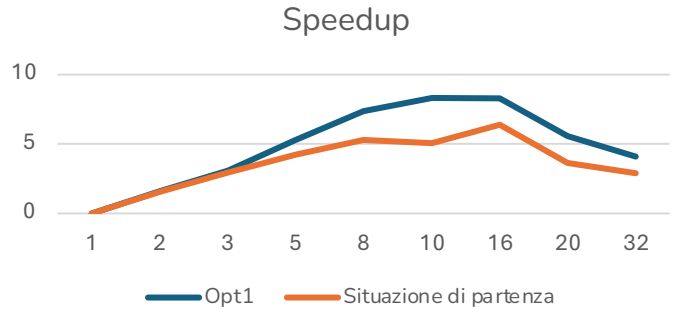
- Preview_State
- Broadcast
- CLOG_Buffer_write2disk
- Receive
- Send

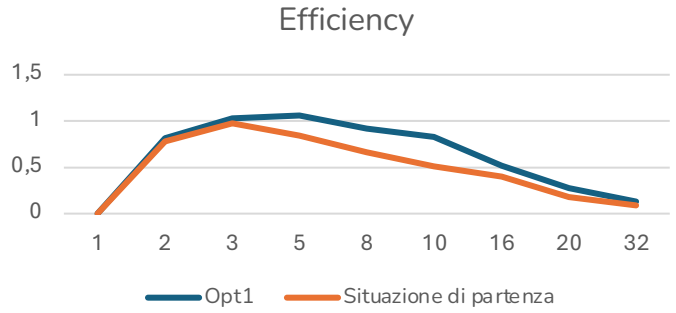


1ª ottimizzazione

Ottimizzazione nel punto Worker 3.
Il Worker non invia più tre messaggi (userA, userU e correlazione) ma, attraverso un MPI_DATA_TYPE, invia un unico messaggio contenente le 3 informazioni, che servono, in ogni caso, contemporaneamente al master.

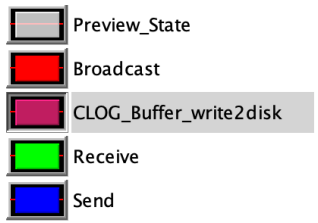
Opt1					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,624	0,812	9,17
1	3	3	3,089	1,030	4,82
1	5	5	5,299	1,060	2,81
1	8	8	7,371	0,921	2,02
1	10	10	8,318	0,832	1,79
1	16	16	8,272	0,517	1,8
2	20	20	5,556	0,278	2,68
2	32	32	4,102	0,128	3,63

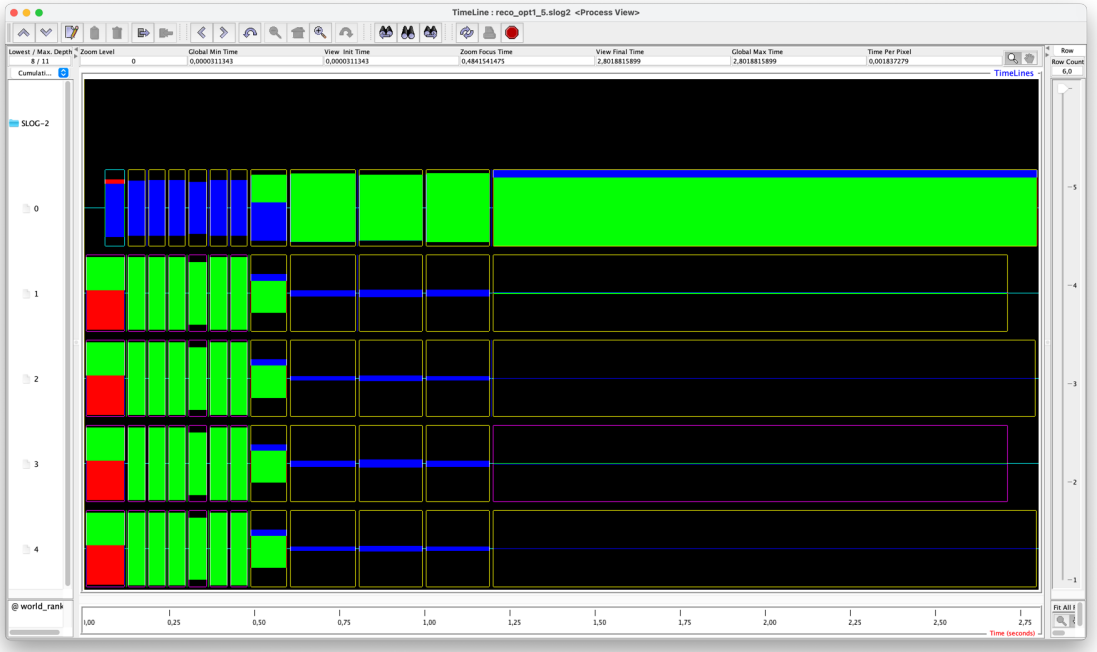




1ª ottimizzazione

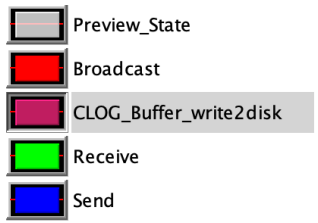
Ottimizzazione nel punto Worker 3.
Il Worker non invia più tre messaggi (userA, userU e correlazione) ma, attraverso un MPI_DATA_TYPE, invia un unico messaggio contenente le 3 informazioni, che servono, in ogni caso, contemporaneamente al Master.

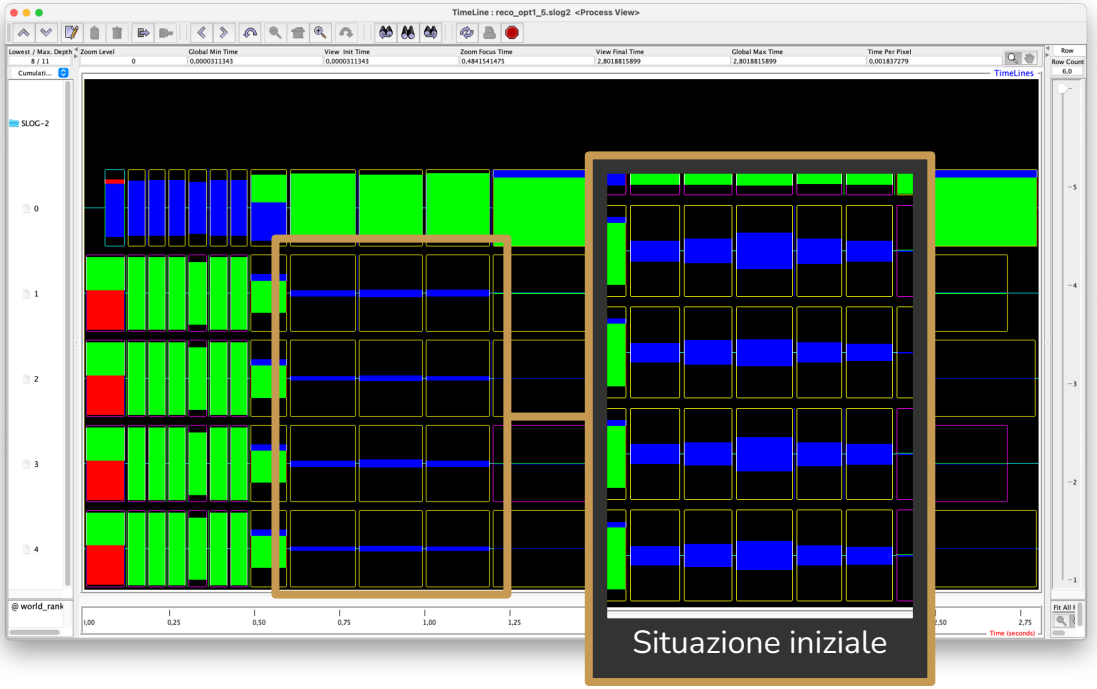




1ª ottimizzazione

Ottimizzazione nel punto Worker 3.
Il Worker non invia più tre messaggi (userA, userU e correlazione) ma, attraverso un MPI_DATA_TYPE, invia un unico messaggio contenente le 3 informazioni, che servono, in ogni caso, contemporaneamente al Master.



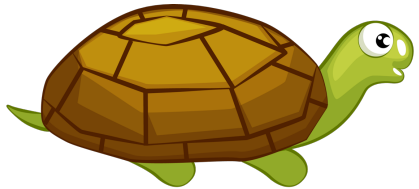


2ª ottimizzazione

Inizializzazione simmetrica

Il tentativo prevedeva che **ogni processo Worker** leggesse **tutta** la matrice da **file**, nonché i parametri di configurazione ricevuti tramite **riga di comando**, anziché ricevere tali informazioni tramite scambio di messaggi con il **Master**.

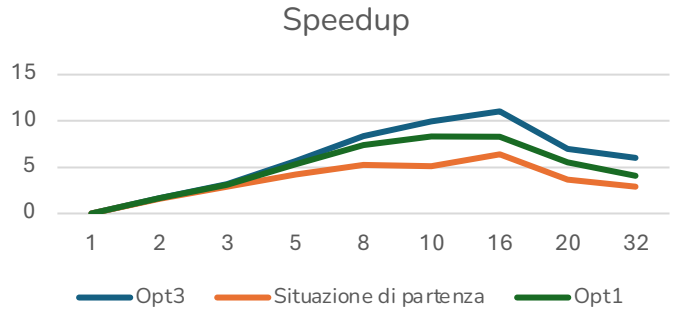
Il tentativo, tuttavia, è risultato **fallimentare** comportando un degrado di prestazioni notevole, in quanto, probabilmente, la **velocità di accesso al disco** è di ordini di grandezza **inferiori** rispetto allo scambio di messaggi su un singolo nodo.

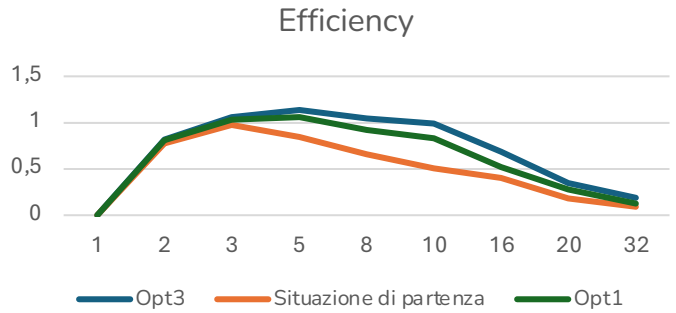


3ª ottimizzazione

Ottimizzazione nel punto Master 2.
Il **Master** non invia più due messaggi (userA, userU) ma, attraverso un MPI_DATA_TYPE, invia un **unico messaggio** contenente le 2 informazioni, che servono, in ogni caso, contemporaneamente al **Worker**.

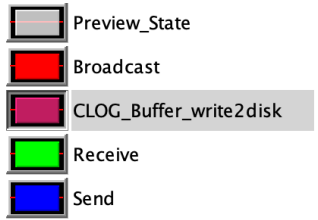
Opt3					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,631	0,815	9,13
1	3	3	3,182	1,061	4,68
1	5	5	5,683	1,137	2,62
1	8	8	8,365	1,046	1,78
1	10	10	9,927	0,993	1,5
1	16	16	11,030	0,689	1,35
2	20	20	6,991	0,350	2,13
2	32	32	6,004	0,188	2,48

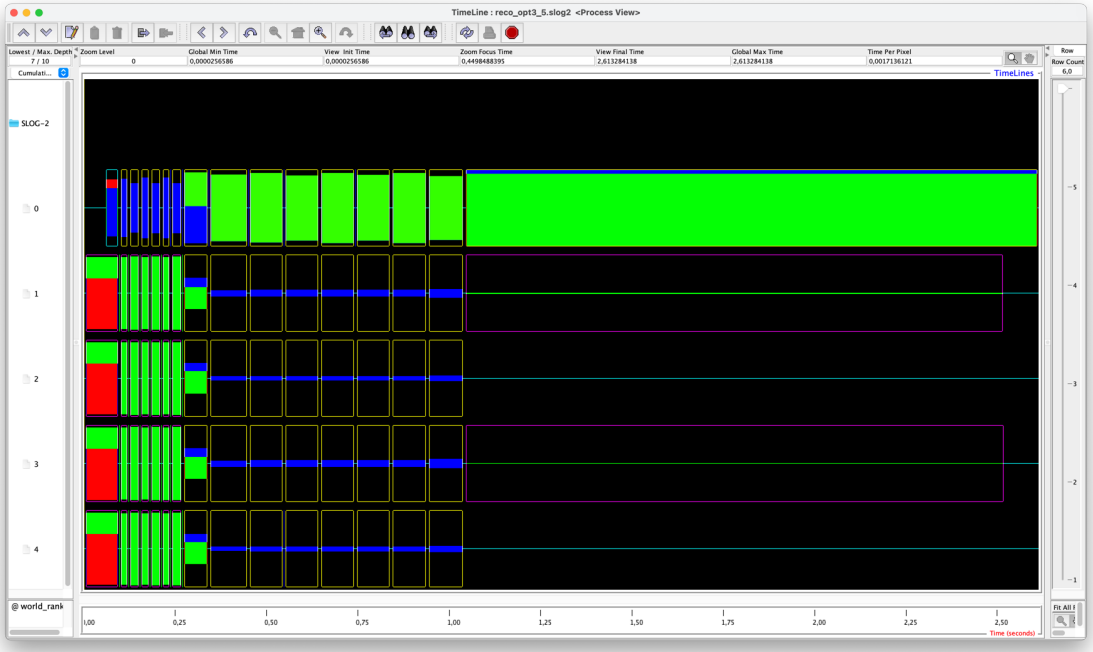




3ª ottimizzazione

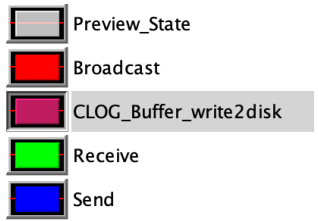
Ottimizzazione nel punto Master 2.
Il **Master** non invia più due messaggi (userA, userU) ma, attraverso un MPI_DATA_TYPE, invia un **unico messaggio** contenente le 2 informazioni, che servono, in ogni caso, contemporaneamente al **Worker**.

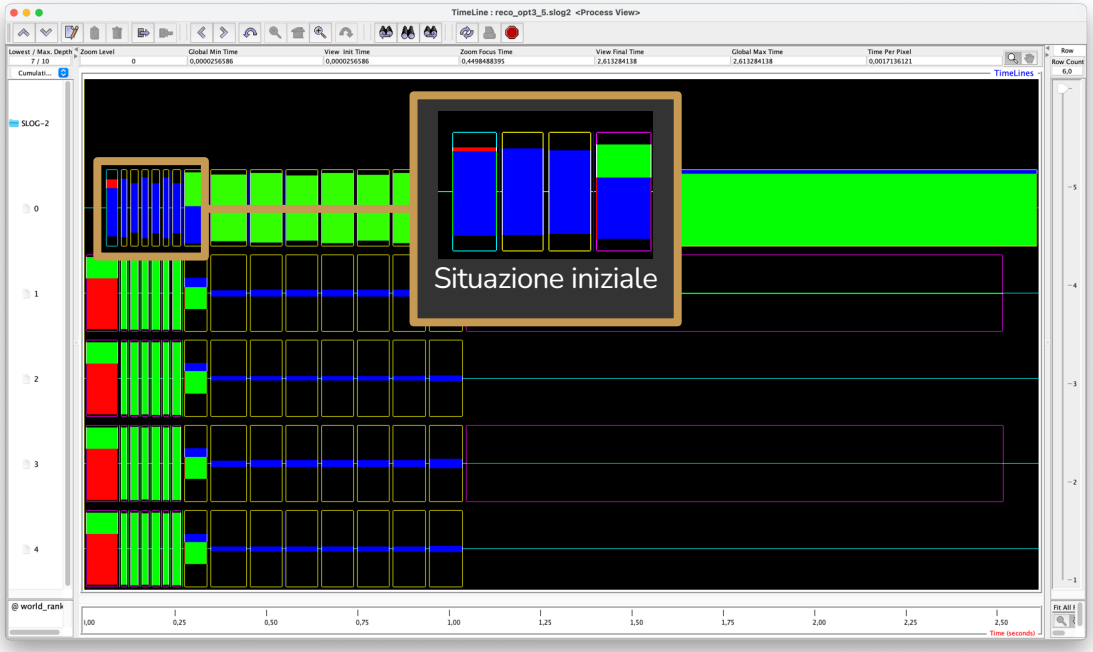




3ª ottimizzazione

Ottimizzazione nel punto Master 2.
Il **Master** non invia più due messaggi (userA, userU) ma, attraverso un MPI_DATA_TYPE, invia un **unico messaggio** contenente le 2 informazioni, che servono, in ogni caso, contemporaneamente al **Worker**.



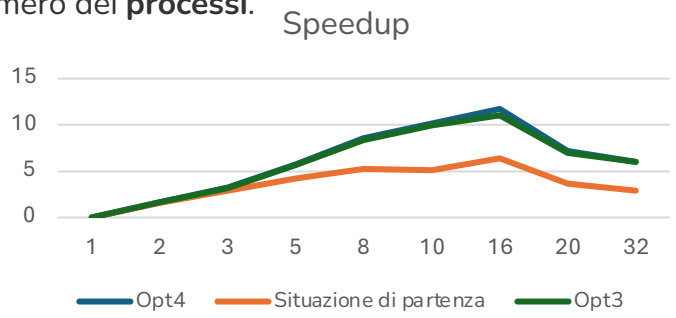


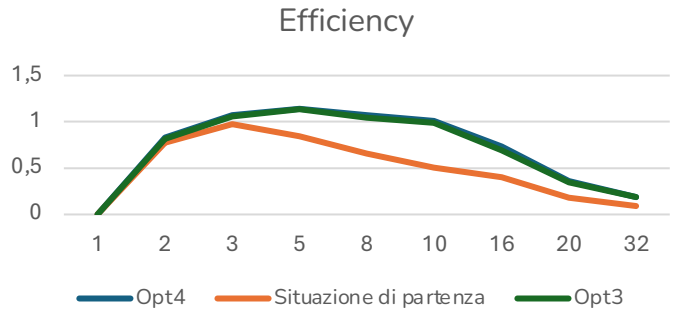
4ª ottimizzazione

Ottimizzazione nel punto Master 2.
Conoscendo il numero di messaggi che riceve ogni Worker, il **Master** non invia più il valore sentinella «-1», riducendo contemporaneamente sia il numero delle **send**, che, dualmente, il numero delle **receive**.
Di conseguenza, il numero di messaggi risparmiati è pari al numero dei **processi**.

```
459         int n_pairs = (users* (users-1))/2;
460         int n_receive = n_pairs/numWorkers;
461         if(taskId<= (n_pairs)%numWorkers){
462             n_receive++;
463         }
```

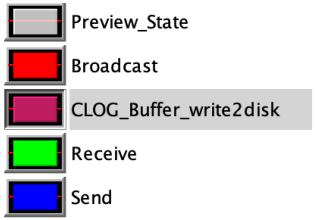
Opt4					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,664	0,832	8,95
1	3	3	3,223	1,074	4,62
1	5	5	5,705	1,141	2,61
1	8	8	8,557	1,070	1,74
1	10	10	10,129	1,013	1,47
1	16	16	11,724	0,733	1,27
2	20	20	7,193	0,360	2,07
2	32	32	6,004	0,188	2,48

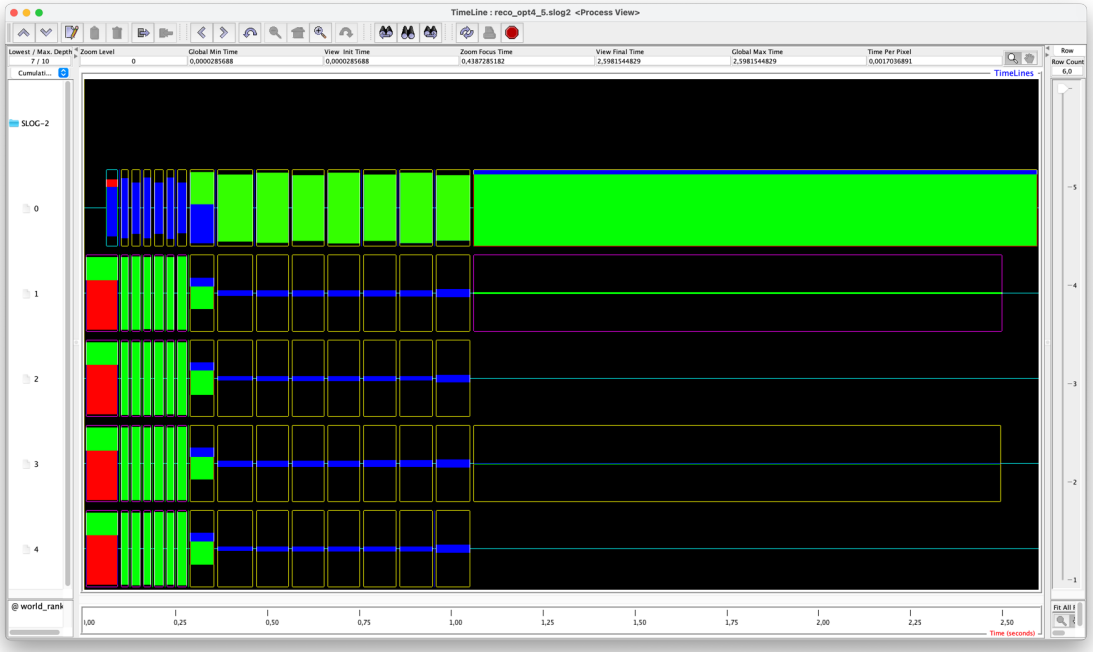




4ª ottimizzazione

Ottimizzazione nel punto Master 2.
Conoscendo il numero di messaggi che riceve ogni Worker, il **Master** non invia più il valore sentinella «-1», riducendo contemporaneamente sia il numero delle **send**, che, dualmente, il numero delle **receive**.
Di conseguenza, il numero di messaggi risparmiati è pari al numero dei **processi**.



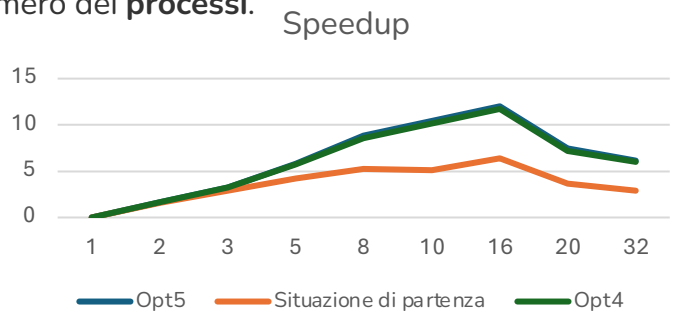


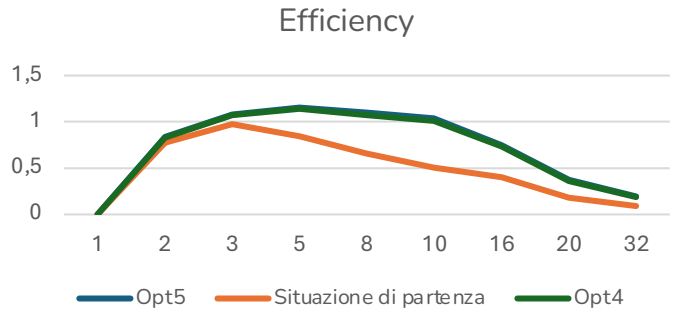
5ª ottimizzazione

Ottimizzazione nel punto Master 4.
Conoscendo il numero di messaggi che riceve ogni Worker, il **Master** non invia più il valore sentinella «-1», riducendo contemporaneamente sia il numero delle **send**, che, dualmente, il numero delle **receive**.
Di conseguenza, il numero di messaggi risparmiati è pari al numero dei **processi**.

```
237         int n_receive = users / numWorkers;
238         if(taskId<=users % numWorkers){
239             n_receive++;
240         }
```

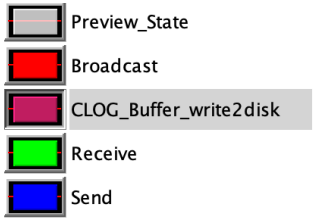
Opt5					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,671	0,836	8,91
1	3	3	3,244	1,081	4,59
1	5	5	5,771	1,154	2,58
1	8	8	8,811	1,101	1,69
1	10	10	10,413	1,041	1,43
1	16	16	12,008	0,751	1,24
2	20	20	7,445	0,372	2
2	32	32	6,178	0,193	2,41

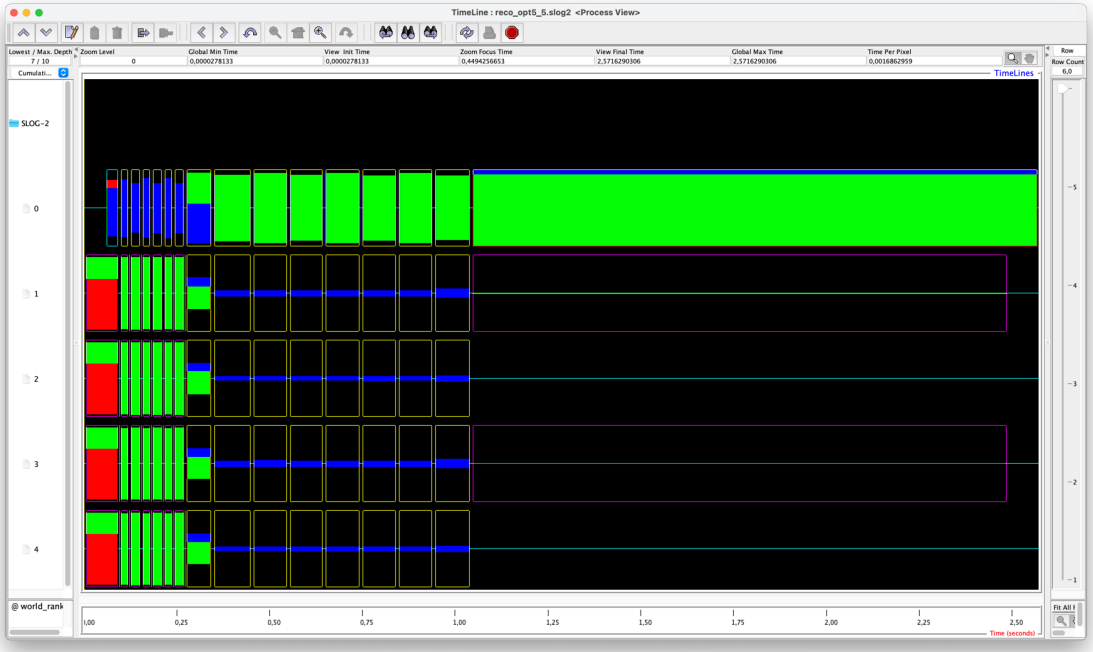




5ª ottimizzazione

Ottimizzazione nel punto Master 4.
Conoscendo il numero di messaggi che riceve ogni Worker, il **Master** non invia più il valore sentinella «-1», riducendo contemporaneamente sia il numero delle **send**, che, dualmente, il numero delle **receive**.
Di conseguenza, il numero di messaggi risparmiati è pari al numero dei **processi**.

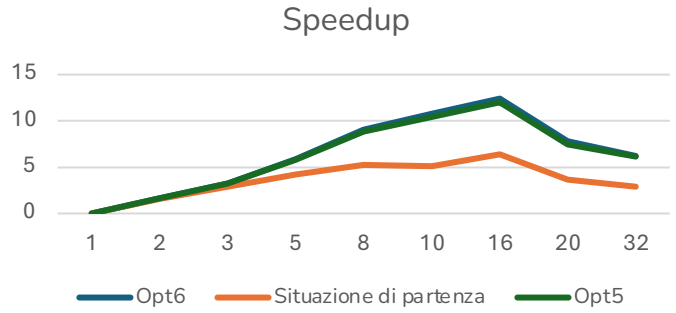


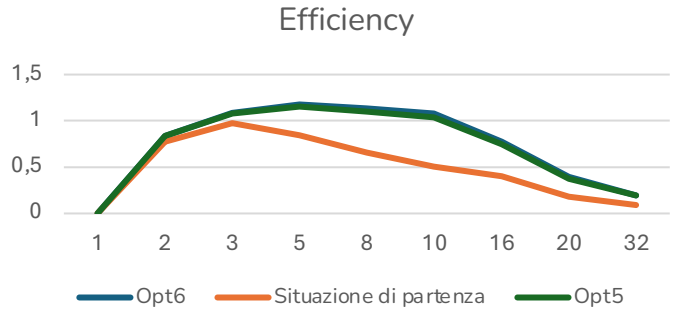


6ª ottimizzazione

Passaggio da **comunicazione** sincrona ad **asincrona**.
Ci aspettavamo miglioramenti più netti, tuttavia **c'è poca concorrenza tra Master e Worker**, in quanto bisogna attendere i risultati intermedi di tutti i Worker prima che il Master possa procedere al passaggio successivo.

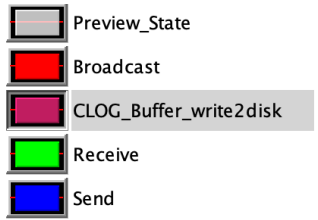
Opt6					
Nodi	**Processi**	**CPU**	**Speedup**	**Efficiency**	**Exec_Time**
1	1	1	0	0	14,89
1	2	2	1,671	0,836	8,91
1	3	3	3,265	1,088	4,56
1	5	5	5,885	1,177	2,53
1	8	8	9,079	1,135	1,64
1	10	10	10,790	1,079	1,38
1	16	16	12,408	0,776	1,2
2	20	20	7,837	0,392	1,9
2	32	32	6,204	0,194	2,4

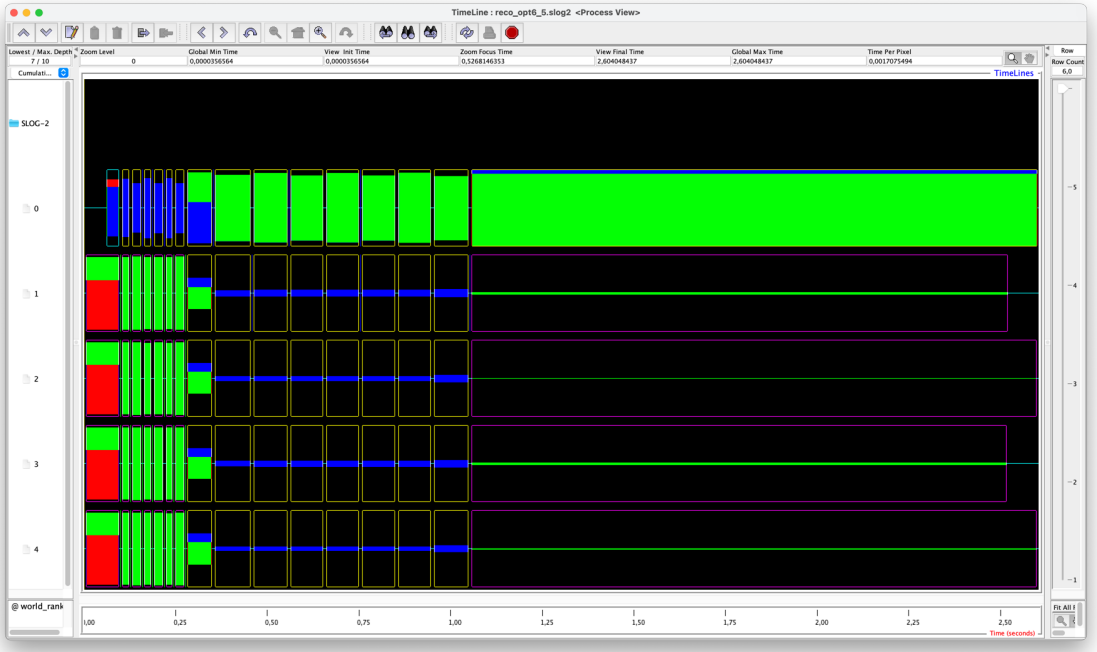




6ª ottimizzazione

Passaggio da **comunicazione** sincrona ad **asincrona**.
Ci aspettavamo miglioramenti più netti, tuttavia **c'è poca concorrenza tra Master e Worker**, in quanto bisogna attendere i risultati intermedi di tutti i Worker prima che il Master possa procedere al passaggio successivo.





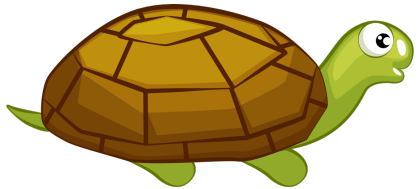
7ª ottimizzazione

Aumento della concorrenza

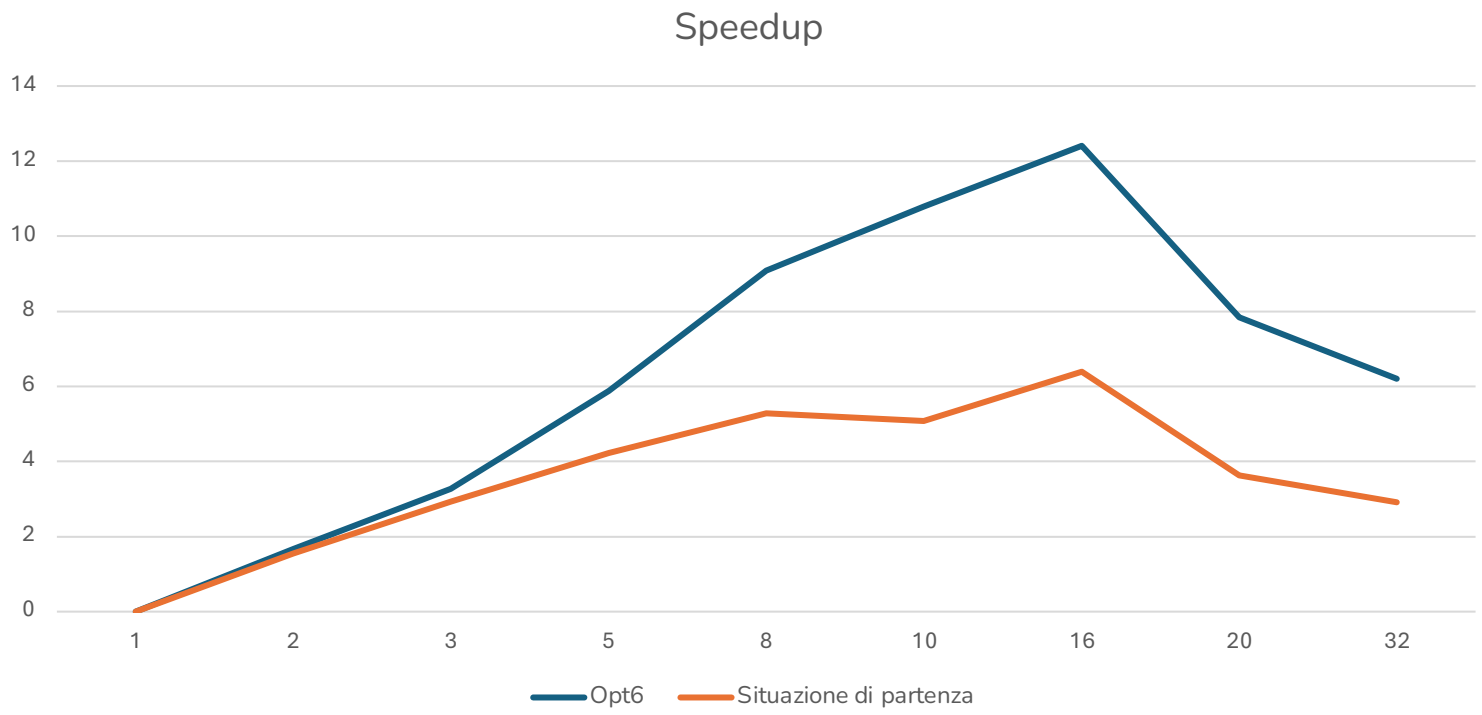
Il tentativo prevedeva che i Worker calcolassero le correlazioni tra una **receive** e un'altra, **senza attendere** la ricezione di tutte le coppie di utenti.

Il tentativo, tuttavia, è risultato **fallimentare** comportando un degrado di prestazioni notevole, in quanto, probabilmente, esistendo un meccanismo di invio/ricezione Round Robin anche di tale informazione, si verifica una situazione di stallo in attesa che arrivi il proprio turno per inviare/ricevere il risultato.

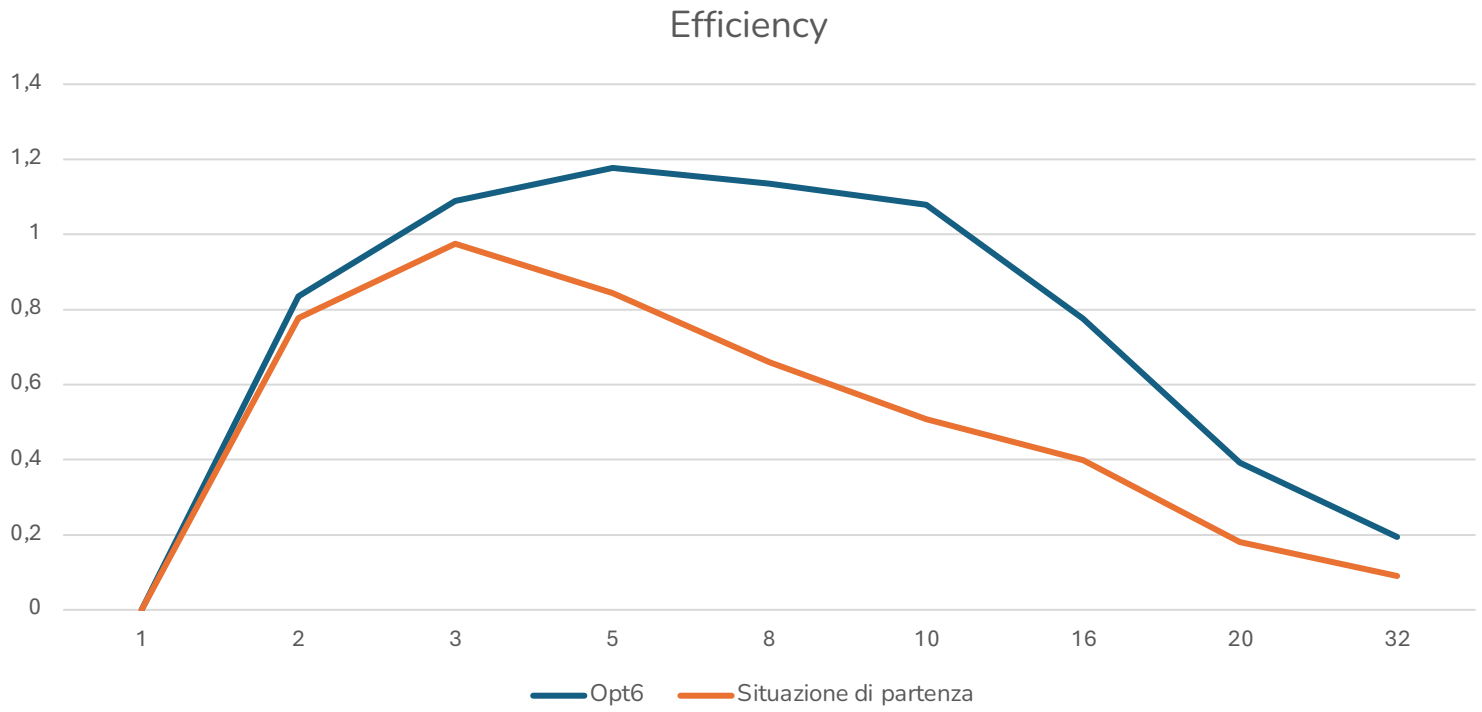
Si può pensare, come sviluppo futuro, di lavorare sul pattern di comunicazione tra i processi.



Comparazione finale



Comparazione finale



Comparazione finale

			Situazione di partenza			Opt6		
Nodi	Processi	CPU	Speedup	Efficiency	Exec_Time	Speedup	Efficiency	Exec_Time
1	1	1	0	0	14,89	0	0	14,89
1	2	2	1,554	0,777	9,58	1,671	0,836	8,91
1	3	3	2,925	0,975	5,09	3,265	1,088	4,56
1	5	5	4,218	0,844	3,53	5,885	1,177	2,53
1	8	8	5,280	0,660	2,82	9,079	1,135	1,64
1	10	10	5,082	0,508	2,93	10,790	1,079	1,38
1	16	16	6,391	0,399	2,33	12,408	0,776	1,2
2	20	20	3,623	0,181	4,11	7,837	0,392	1,9
2	32	32	2,908	0,091	5,12	6,204	0,194	2,4

Repository

https://github.com/Pi0Ant0ni0/recommendationSystemMPI_CPAP